Informatikwerkstatt, Foliensatz 11
Motorsteuerung

G. Kemnitz

Institut für Informatik, TU Clausthal (IW-F11)

6. Dezember 2021

Inhalt:

Interaktive Übungen:

1. Drehzahlsteuerung (test_pwm)
2. Messung der Umdrehungsgeschwindigkeit (rotmess)

Themen, die fortgesetzt werden:

- PWM-Erzeugung mit Timern.
- Treiberprogrammierung mit ISR.

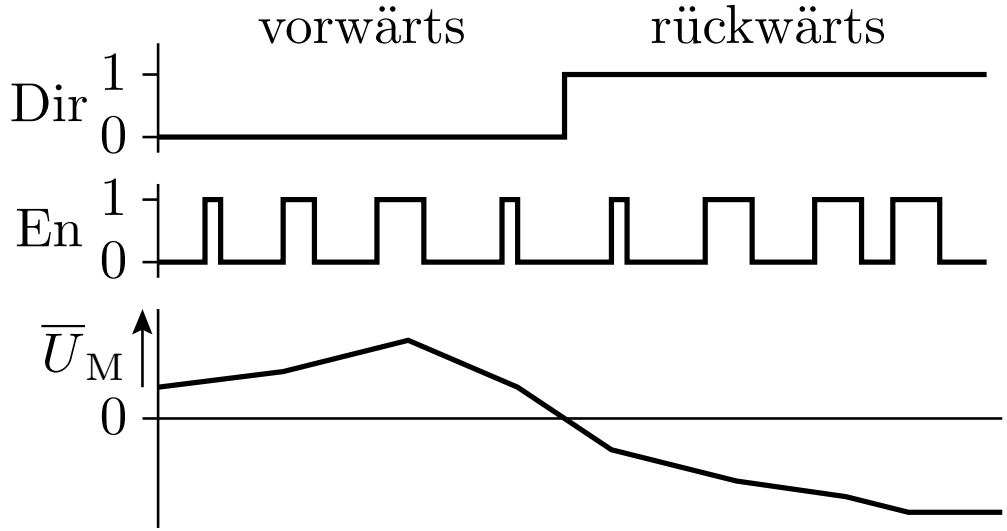
Drehzahlsteuerung

Prinzip und Motortest

Drehzahlsteuerung durch Pulsweitenmodulation

Pulsweitenmodulation (PWM) schaltet die Motoren schnell ein und aus. Drehzahlsteuerung über die relative Einschaltzeit.

En	1	0	0	1
Dir	0	0	1	1
	+ M −	M	− M +	

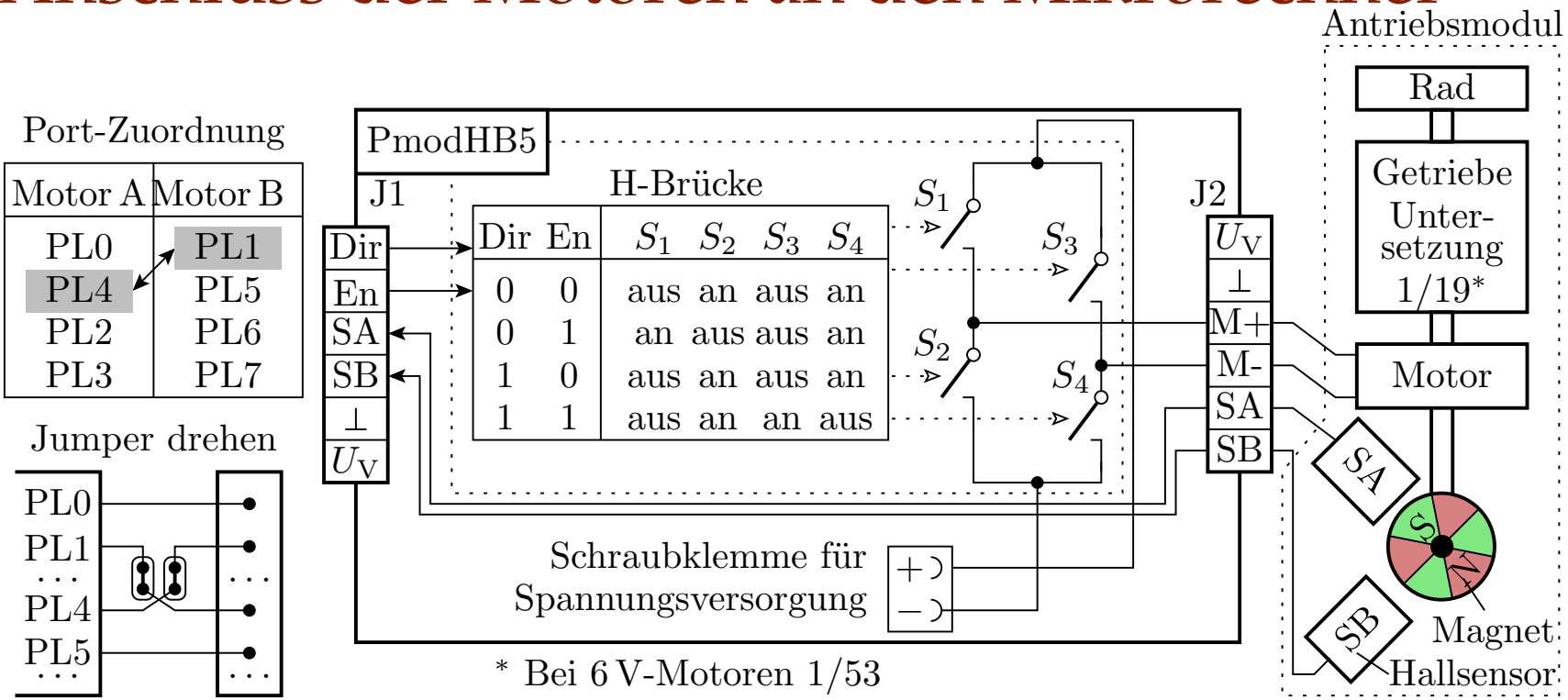


An den Antriebsbaugruppen erfolgt die Einstellung

- der Drehrichtung über ein Richtungsbit Dir und
- der relativen Pulsbreite mit dem En- (Enable-) Signal.

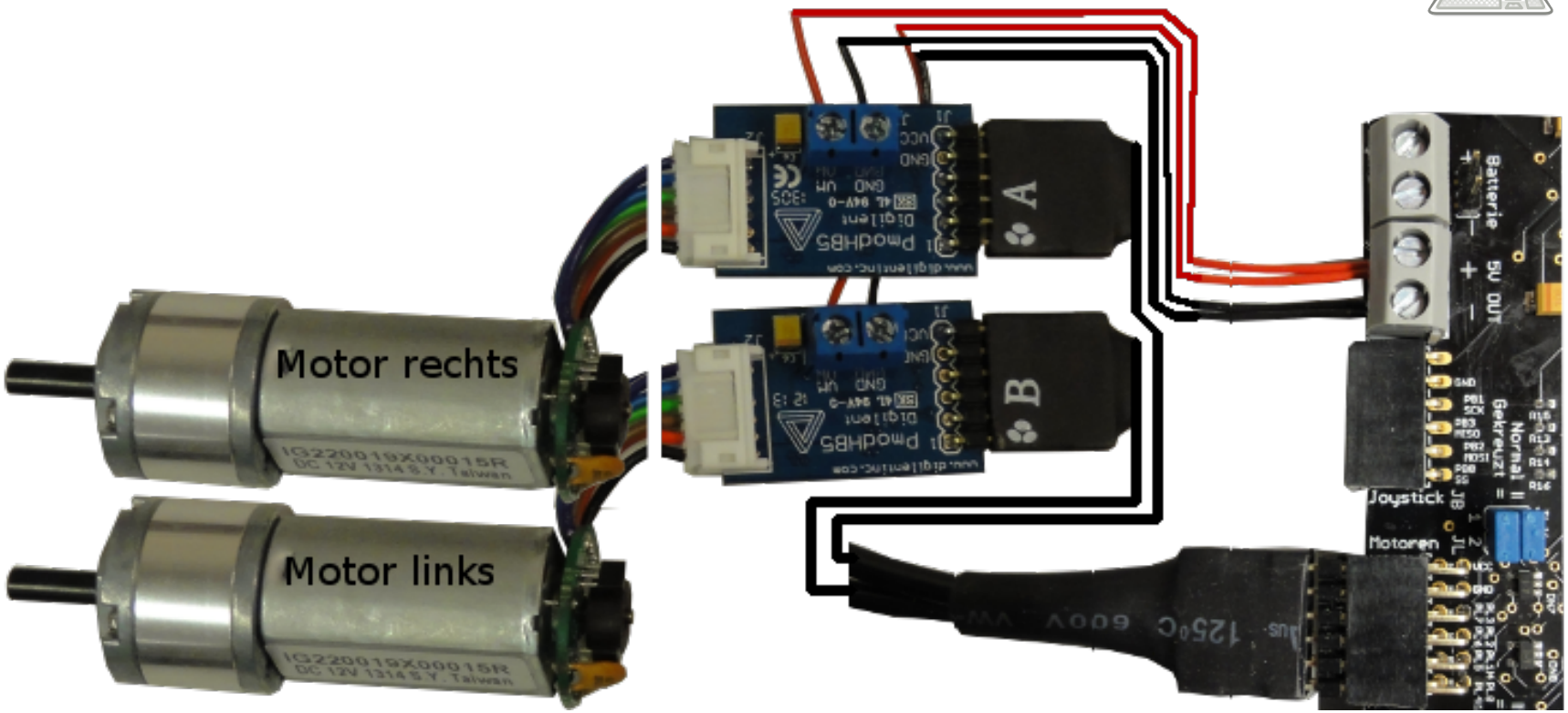
Achtung: Der Wert von Dir darf nur bei EN=0 geändert werden!

Anschluss der Motoren an den Mikrorechner



- Antriebsmodule: Motor, Untersetzungsgetriebe, rotierender Magnet + Hallsensoren zum Zählen der Winkelschritte.
- PmodHB5: H-Brücke, angesteuert über Dir und En. Rückgabe der Hallsensorsignale an den Mikrorechner.

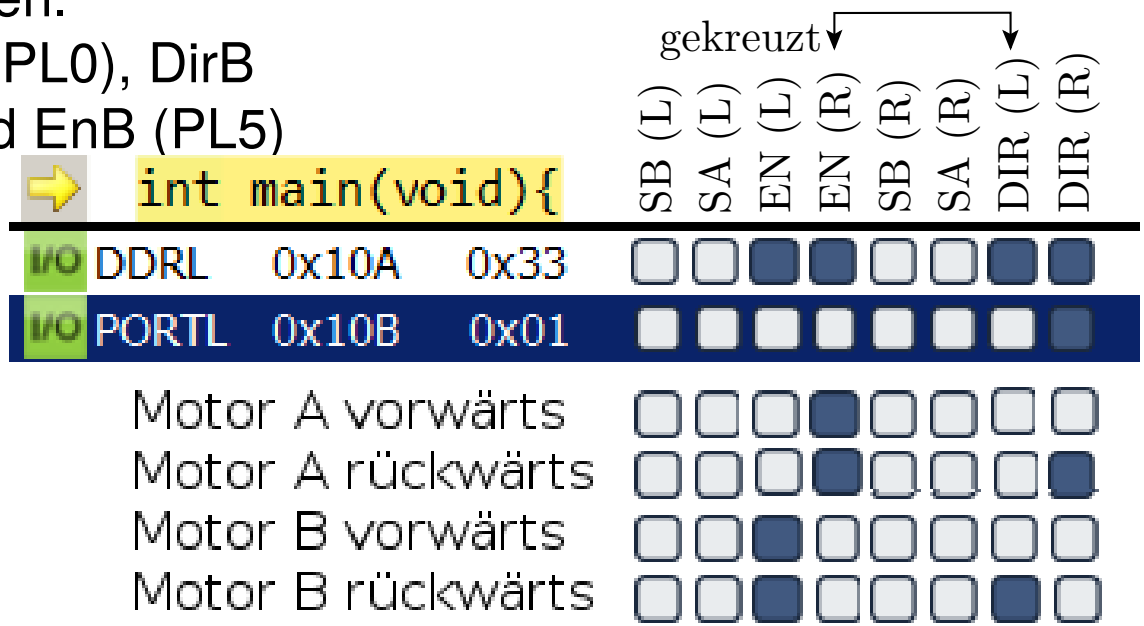
Praktischer Aufbau



- 2×H-Brücke PmodHB5 über Y-Kabel an JL,
- Motoren an die H-Brücken stecken,
- JLX »gekreuzt (=)« (Pin-Tausch PL0 und PL4),
- Spannungsversorgungsdrähte zuschneiden und anschrauben.

Motoren ausprobieren

- Beliebiges Projekt im Debugger starten  . Anhalten.
- I/O > Port L aufklappen.
- Zum Motortest DirA (PL0), DirB (PL1), EnA (PL4) und EnB (PL5) auf Ausgang und Ausgabewerte setzen.
  DIR nur bei EN=0 ändern!

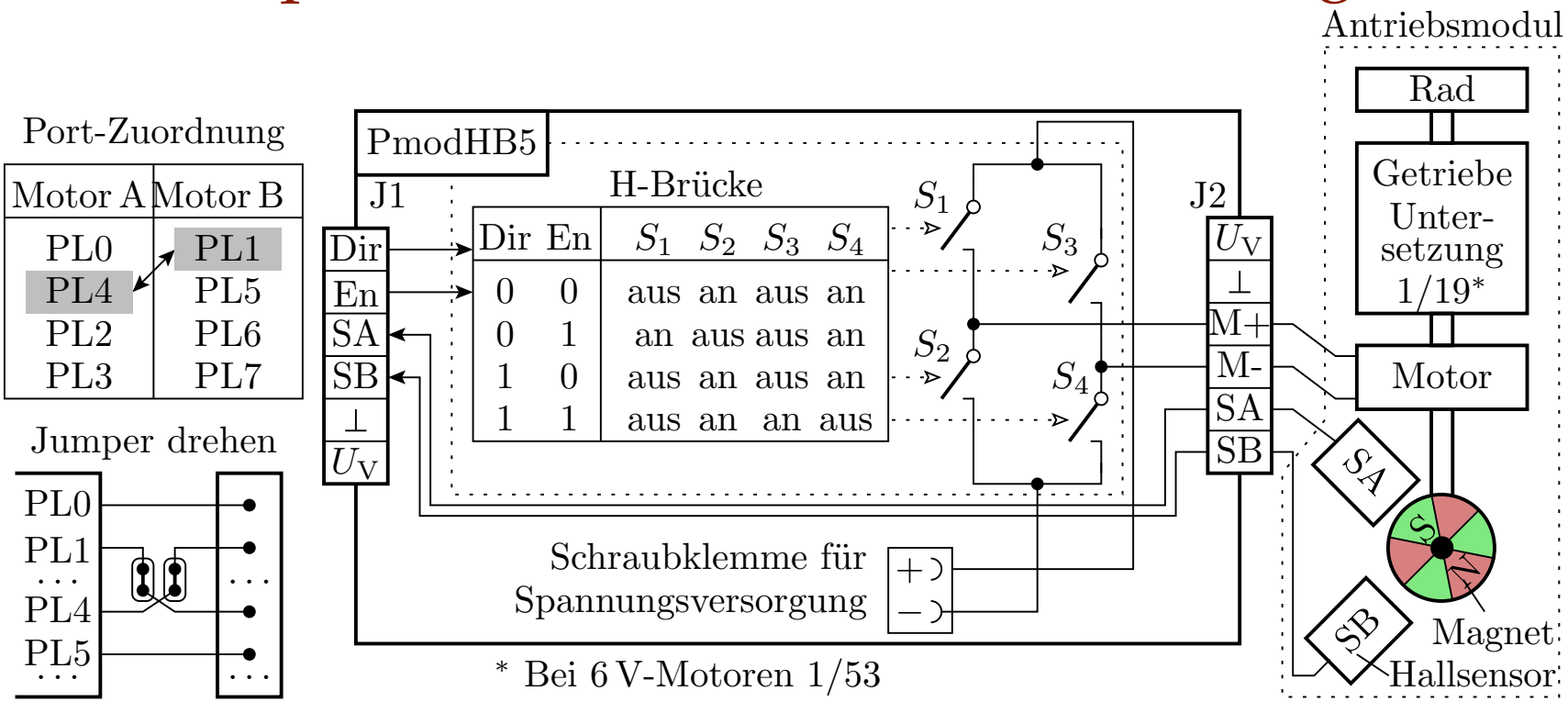


- Motoren vor- und rückwärts drehen lassen.
- Kontrolle der Sensorausgaben mit Multimeter[1].

[1] Die Anzeige von »PINL« wird nur bei Programm-Start-Stop aktualisiert.

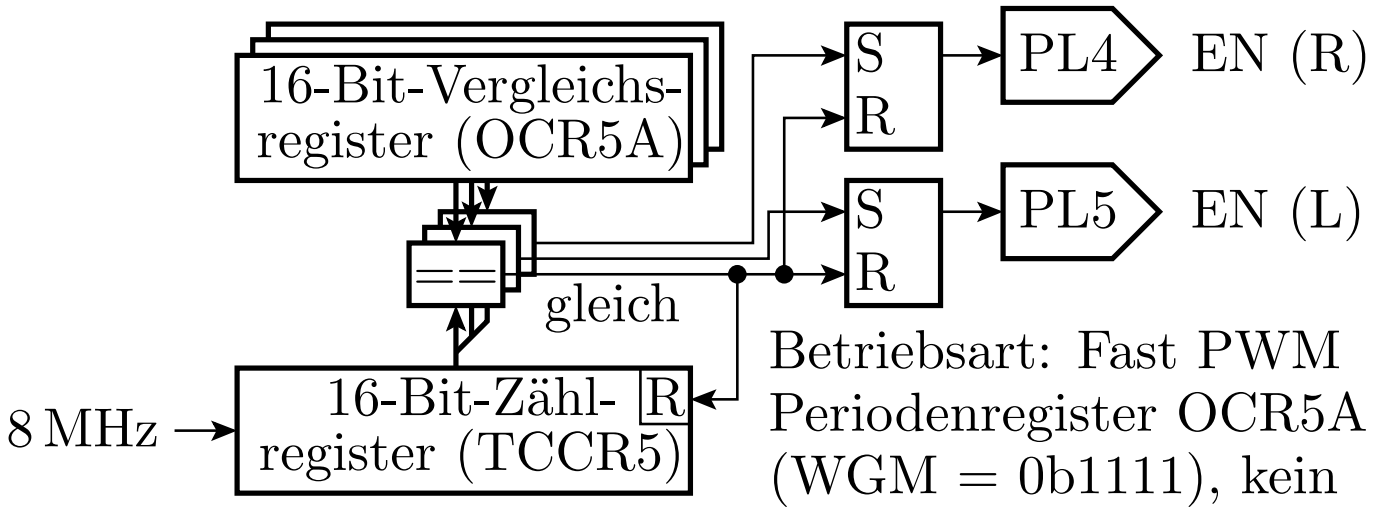
Treiber »pwm«

Treiber »pwm« für die Drehzahlsteuerung

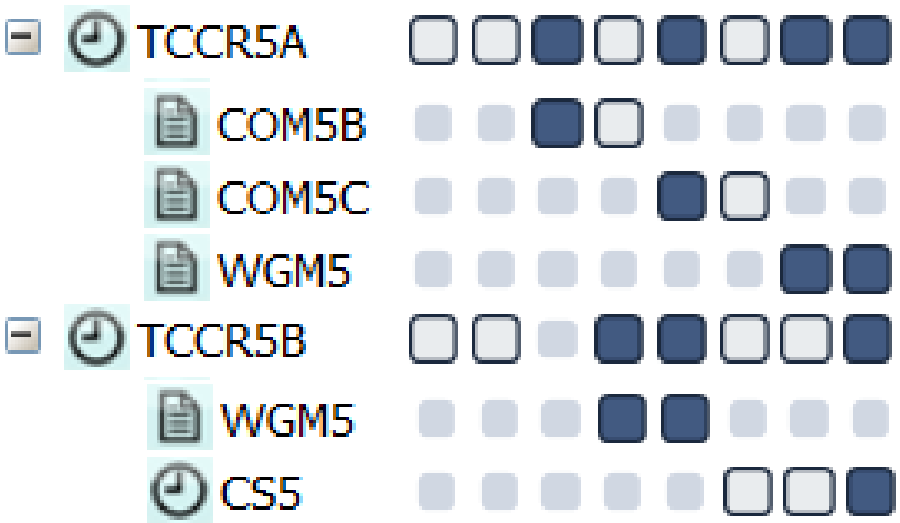


- Der Treiber erwartet die dargestellte Hardware und erzeugt die Dir- und En-Signale für beide Motoren.
- Die gepulsten En-Signale generiert Timer 5 im PWM-Modus ohne ISR an PL4 und PL5.

Timer-Einstellung für die Enable-Signale



Betriebsart: Fast PWM
Periodenregister OCR5A
(WGM = 0b1111), kein
Taktvorteiler (CS=0b001)
PWM-Ausgabe ”Clear on
Compare” (COM5B/C=0b10)



Funktionen des Treibers

- Keine privaten Daten.
- Initialisierungsfunktion.
- Keine ISR oder Schrittfunktion.
- Jeweils eine Funktion für Stopp und Start beider Motoren.
- Jeweils eine Funktion zur Einstellung der Pulsbreite.

Initialisierungsfunktion:

```
void pwm_init(){
 DDRL  =0b00110011;  // EN und DIR als Ausgänge
 pwm_stop();         // Zähltakt und PWM aus ...
 TCNT5 = 0;          // Zählregister löschen
 OCR5A = 0x2000;     // Periodenregister (ca. 1ms)
 OCR5B = 0;          // Motor R: Pulsbreite 0
 OCR5C = 0;          // Motor L: Pulsbreite 0
}
```

Stoppfunktion für beide Motoren: Zähltakt und PWM-Ausgabe aus.

```
void pwm_stop(){
  TCCR5A = 0;   //PWM ausschalten
  TCCR5B = 0;   //Zähltakt aus
  PORTL  = 0;   //Enable (Motoren) ausschalten
}
```

Startfunktion für beide Motoren: Zähltakt und PWM-Ausgabe ein.

```
void pwm_start(){
  //COM5B/C=0b10 (PWM-Ausgänge ein)
  TCCR5A = 0b00101011;
  //WGM=0b1111 CS=0b001 (Takt ein)
  TCCR5B = 0b00011001;
}
```

Übergabe der Pulsbreite für den rechten Motor:

```
void pwm_set_R(int16_t pwm){
  if (pwm>=0){
    OCR5B =pwm;
    PORTL |=1;          //DIR-Bit (PL0) setzen
  }
  else{
    OCR5B = -pwm;
    PORTL &= ~1;        //DIR-Bit (PL0) löschen
  }
}
```

- Der Geschwindigkeitswert ist 16-Bit vorzeichenbehaftet.
- Bei Betragswerten größer Periodenwert bleibt das Freigabesignal dauerhaft an.
- In der Funktion für den linken Motor

  ```
  void pwm_set_L(int16_t pwm);
  ```

  ist »OCR5B« durch »OCR5C« und »PL0« durch »PL1« zu ersetzen.

Treibertest

Das Testprogramm

Das Testbeispiel nutzt außer »pwm.h«:

```
#include "comir_pc.h"  //PC-Eingabe
#include "comir_tmr.h" //Bewegungsdauer
```

In »comir_pc.h« sind die Puffergrößen geändert auf:

```
#define COM_PC_RMSG_LEN 6 //Empfang 6 Byte
#define COM_PC_SMSG_LEN 0 //keine Sendenachricht
```

Das Hauptprogramm:

```
uint8_t msg[COM_PC_RMSG_LEN];
int main(void){
  int16_t pwm; uint16_t time;
  com_pc_init();  // Init. PC-Kommunikation
  pwm_init();     // Init. Motor-Treiber
  tmr_init();     // Init. Timer-Treiber
```

- In der Enlosschleife wird auf eine 6-Byte-Nachricht gewartet.
- Wenn sie eintrifft, werden die PWM-Werte gesetzt, der Timer und die Bewegung gestartet.
- Nach der Wartezeit wird der Motor ausgeschaltet.

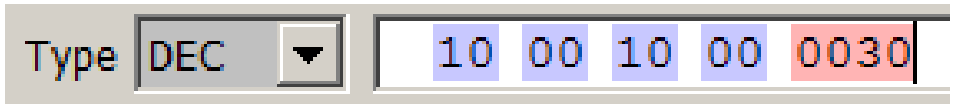
```
sei();                      //Interrupts global ein
while(1){
 if (com_pc_get(msg)){ //wenn neue Nachricht
  pwm = msg[0]<<8 | msg[1];
  pwm_set_R(pwm);           //PWM-Wert für Motor R
  pwm = msg[2]<<8 | msg[3];
  pwm_set_L(pwm);           //PWM-Wert für Motor L
  time = msg[4]<<8 | msg[5];
  tmr_start(time, 0);   //Timer Kanal 0 starten
  pwm_start();              //PWM (Motoren) starten
 }
 if (!tmr_restzeit(0)) //wenn Timer abgelaufen
  pwm_stop();               //PWM und Motoren aus
 }
}
```

Treiber »pwm« ausprobieren

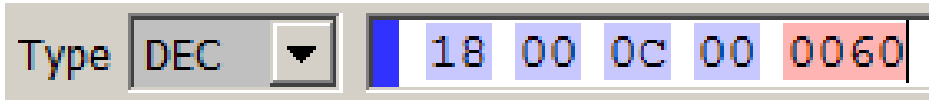
- siehe Folie 7.
- PmodUSBUSART an JH oben und USB-Verbindung zum PC.
- JHX und JLX auf »gekreuzt (=)« .
- Projekt »F11-test_pwm\test_pwm« übersetzen und starten.
- HTerm starten. 8N1 9600 Baud. Com Auswahl. Connect.

Testbeispiele:

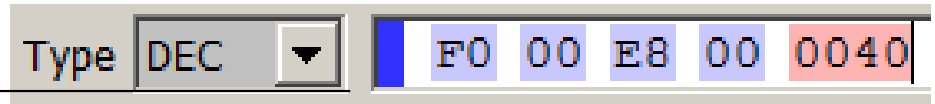
- Motoren R und L mit 50% für 3 s vorwärts[2]:



- Motor R mit 75% und Motor L mit 37,5% für 6 s vorwärts:



- Motor R mit 50% und Motor L 75% rückwärts für 4 s.



[2] In den Bildern blaue Eingaben »HEX« und rote Eingaben »DEC«.

Erstellung weiterer Testbeispiele

- Die Motoren werden mit 6-Byte-Nachrichten $B_0 B_1 \ldots B_5$ (B_i – Byte i) angesteuert.
- Byte B_0 und B_1 definieren die relative Pulsbreite Motor R:

$$\eta_{\mathrm{R}} = \begin{cases} 1 & B_0 \geq 0\mathrm{x}20 \\ \frac{|16 \cdot B_0 + B_1|}{0\mathrm{x}2000} & B_0 < 0\mathrm{x}20 \end{cases}$$

- Byte B_2 und B_3 definieren die relative Pulsbreite Motor L:

$$\eta_{\mathrm{L}} = \begin{cases} 1 & B_2 \geq 0\mathrm{x}20 \\ \frac{|16 \cdot B_2 + B_3|}{0\mathrm{x}2000} & B_2 < 0\mathrm{x}20 \end{cases}$$

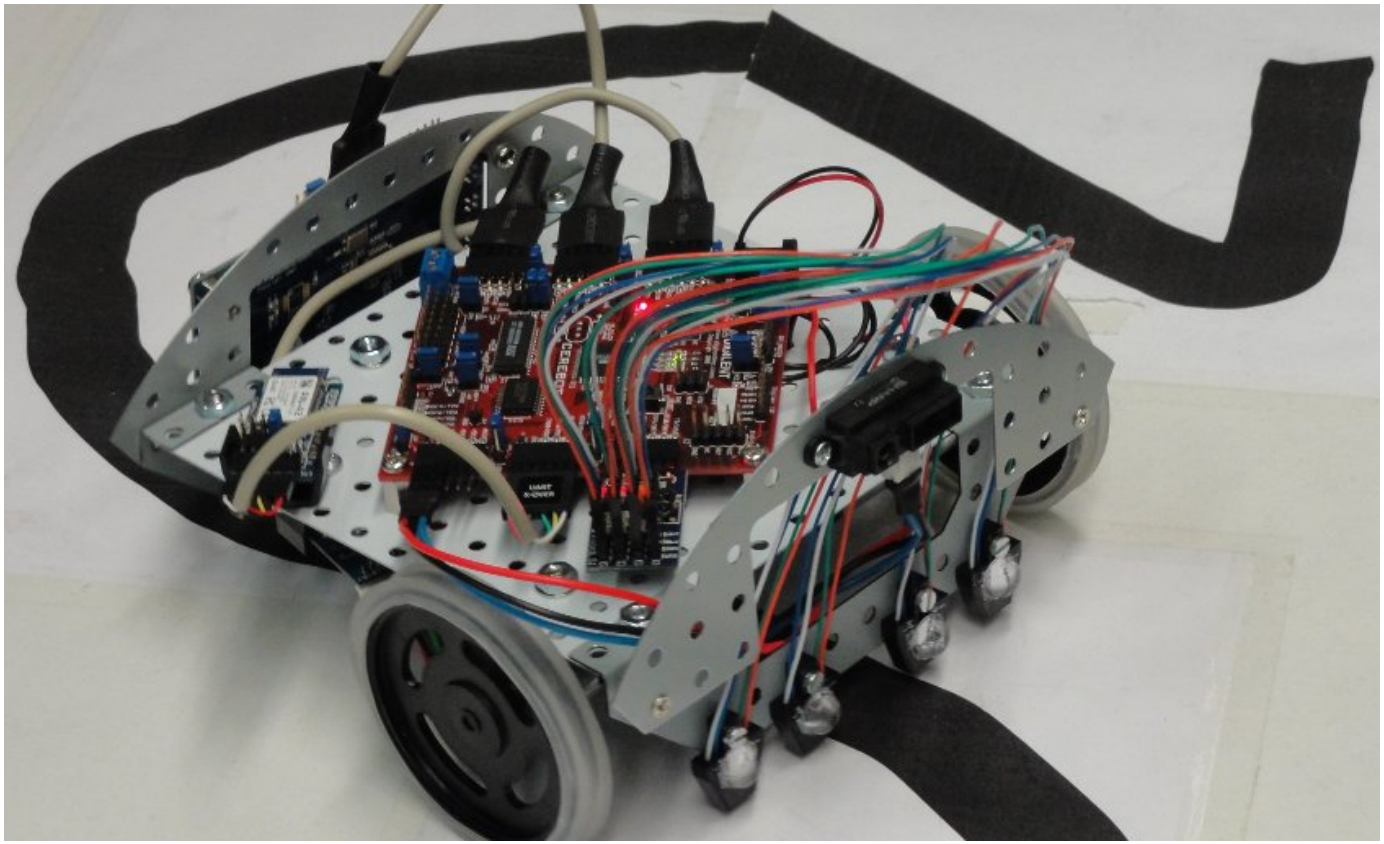
- Byte B_4 und B_5, auch zusammen als Dezimalzahl eingebbar, definieren die Bewegungsdauer:

$$t = \frac{16 \cdot B_3 + B_4}{10}\,\mathrm{s}$$

Winkelmessung

Messprinzip

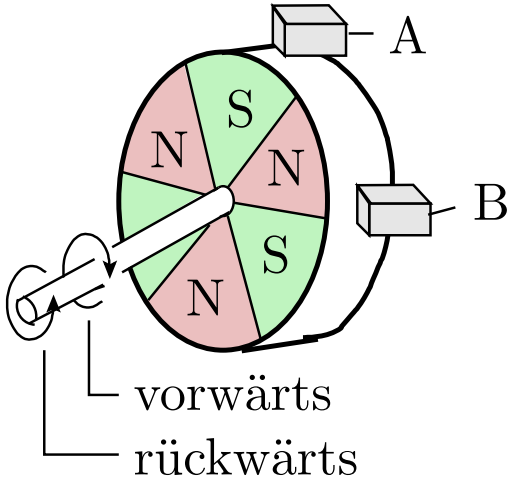
Drehwinkel und Fahrzeugposition

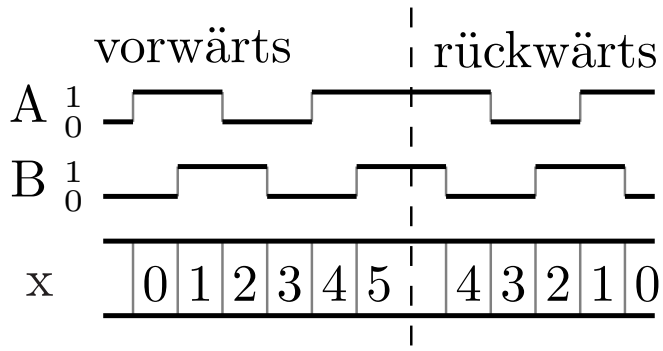


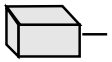
Zählen der Winkelschritte an beiden Antriebsrädern. Erweiterbar zu einer 2D+R-Positionsbestimmung[3].

[3] »2D-Position und Richtung relativ zur Startposition und Ausrichtung.

Winkelschrittzähler





	Hallsensoren
x	Winkel in Zwölftelkreisschritten
A*, B*	Abtastwerte Zeitschritt zuvor

A	B	A*	B*	x
0	0	0	0	—
0	1	0	0	−1
1	0	0	0	+1
0	1	0	1	—
0	0	0	1	+1
1	1	0	1	−1
1	0	1	0	—
1	1	1	0	+1
0	0	1	0	−1
1	1	1	1	—
1	0	1	1	−1
0	1	1	1	+1

- Auflösung 1/12 Motorumdrehungen. Eine Radumdrehung sind 19 Motorumdrehungen. Max. 5 Radumdrehungen / s.
- Die Sensorbitwerte müssen mindestens einmal je Winkelschritt gelesen und verarbeitet werden ($\geq$1200 s^{-1}).

Geschwindigkeit- und 2D-R-Positionsbestimmung

- Geschwindigkeit: Winkelschritte für eine bestimmte Zeit zählen.
- 2D+R-Position: Für jeden Auswerteschritt der Sensorbits 9 Fälle unterscheiden:

Δw_R	0	-1	+1	0	-1	+1	0	-1	+1
Δw_L	0	0	0	-1	-1	-1	+1	+1	+1
Δs	0	$-s_0$	s_0	$-s_0$	$-2s_0$	0	s_0	0	$2s_0$
$\Delta\alpha$	0	$-\alpha_0$	α_0	α_0	0	$2\alpha_0$	$-\alpha_0$	$-2\alpha_0$	0

α – Bewegungsrichtung; Δs – Schrittweite in Richtung α; $s_0 = \frac{\pi \cdot d}{2 \cdot 19 \cdot 12}$ – Basisschrittweite und $\alpha_0 = \tan\left(\frac{2 \cdot s_0}{a}\right)$ – Rotation je Winkelschritt; d – Raddurchmesser; a – Radabstand.

Weiterführung als selbstständig zu lösende Aufgabe 11.4.

Treiber »rotmess«

Der Treiber »rotmess«

Bestimmt die Anzahl der Winkelschritte für ein Zeitintervall, im folgenden Testbeispiel für 1 s in der zyklisch alle 0,5 ms gestarteten Tmr1-ISR:

- Inkrement eines Zeitzählers.
- Einlesen der Sensorbitwerte SA und SB für beide Räder.
- Aus diesen und den vorhergehenden Sensorbitwerten Berechnung der Drehwinkeländerungen $\Delta w \in \{-1, 0, +1\}$.
- Summierung der Δw für je 1 s (2.000 Schritte[4] zu je 0,5 ms).
- Danach werden ein Ereignisbit gesetzt, die Schrittzähler gelöscht und die Zählwerte gesichert.

Eine Get-Funktion liest und löscht die gesicherten Zählwerte.

Später wird die Messdauer von 1 s auf die Dauer eines Reglerschritt von 20 ms verringert.

[4] Im Header »rotmess.h« einstellbar. Später Schrittzeit des Reglers.

Private Daten und Initialisierung

```
int16_t Ct_T;         //Zeitzähler
int16_t Ct_R, Ct_L; //Geschwindigkeitszähler
int16_t speed_R, speed_L;//Geschwindigkeitswerte
int8_t sens_R, sens_L;   //Bit(3:2) neue und Bit
                         //(1:0) alte Sensorwerte
uint8_t rotmess_err_ct;  //Fehlerzähler, nur Debug
uint8_t new_dat;     //0 keine neuen, 1 neue Daten
```

Initialisierungsfunktion:

- Timer 0, CTC-Mode, 0,5 ms Periode, OCR0A-Interrupt[5]:

```
void rotmess_init(){
//Timer 0 für OCR0A-Interupts alle 0.5 ms einrichten
  TCCR0A  = 0b10;  //WGM = 0b010: CTC Mode mit OCR0A
  TCCR0B  = 0b011; //CS  = 0b011 (Vorteiler 64)
  OCR0A   = 62;    //OCR = (0,5 ms*8MHz)/(2^8)-1
  TIMSK0 |= 1<<OCIE0A;//OCR0A-Interrupt freigeben
```

[5] 8-Bit-Timer mit weniger Konfigurationsmöglichkeiten.

- Sensorzustand initialisieren. Zähler löschen:

```
  sens_R=(PINL>>4)&0b1100;//Startwerte der Hall-
  sens_L= PINL    &0b1100;//sensoren lesen
  clear_counter();        //Zähler löschen
}
```

Löschfunktion für die Zähler:

```
void clear_counter(){
  Ct_R = 0; Ct_L  = 0; //Winkelschritt- und
  Ct_T = 0;            //Zeitzähler löschen
}
```

Die ISR setzt für beide Motoren die aktuellen und vorherigen Sensorwerte zu einem 4-Bit-Vektor zusammen, ...

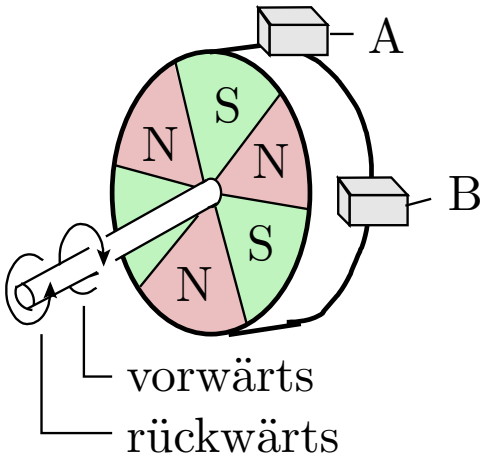
```
ISR(TIMER0_COMPA_vect){
  sens_R = (sens_R>>2) | ((PINL>>4) & 0b1100);
  sens_L = (sens_L>>2) |  (PINL & 0b1100);
```

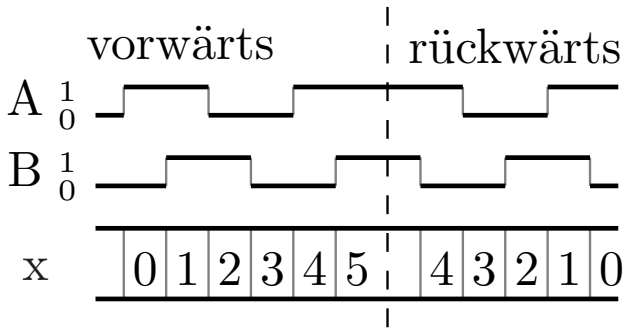
bestimmt mit einer Funktion QuadEnc() den Winkel-Inkrement (WB: $-1, 0, +1$), zählt die Zeit weiter, ...

```
    Ct_R += QuadEnc(sens_R);
    Ct_L += QuadEnc(sens_L);
    Ct_T++;
    if (Ct_T>=ABTASTSCHRITTE){
      speed_R = Ct_R;
      speed_L = Ct_L;
      if (new_dat) rotmess_err_ct++;
      new_dat=1;
      clear_counter();
    }
  }
```

Nach einer als Konstante definierten Anzahl von Abtastschritten werden die Zählwerte gespeichert, ein Flag »neue Daten« gesetzt« und die Zähler gelöscht.

Bestimmung der Winkelbewegung





A	B	A*	B*	x
0	0	0	0	—
0	1	0	0	−1
1	0	0	0	+1
0	1	0	1	—
0	0	0	1	+1
1	1	0	1	−1
1	0	1	0	—
1	1	1	0	+1
0	0	1	0	−1
1	1	1	1	—
1	0	1	1	−1
0	1	1	1	+1

```
int8_t   QuadEnc(uint8_t  sensdat){
 switch (sensdat){
  case  0b0010:
  case  0b0100:
  case  0b1011:
  case  0b1101:
  return  -1;
```

```
  case  0b0001:
  case  0b0111:
  case  0b1000:
  case  0b1110:
  return  1;
```

```
  case  0b0011:
  case  0b0110:
  case  0b1100:
  case  0b1001:
  if (rotmess_err_ct<0xFF)
    rotmess_err_ct++;
 }
 return  0;
}
```

- Funktion zum Lesen der gemessen Winkelschritte:

```
uint8_t rotmess_get(int16_t *spR, int16_t *spL){
 uint8_t tmp = TIMSK0;    //ISR, die dieselben Daten
 TIMSK0 &= ~(1<<OCIE0A); //bearbeitet sperren
 if (new_dat){            //wenn neue Daten
  *spR = speed_R;         //Ergebnisse kopieren
  *spL = speed_L;
  new_dat = 0;          //neue-Daten-Flag löschen
  TIMSK0 = tmp;         //Interrupt wieder freigeben
  return 1;             //Rückkehr mit "neue Daten"
 }                      //sonst
 TIMSK0 = tmp;          //Interrupt wieder freigeben
 return 0;              //Rückkehr ohne neue Daten
}
```

Zur Fehlerbehandlung im übergeordneten Modul gibt es noch eine Abfragefunktion, ob Abtastfehler im Abfrageintervall aufgetreten sind. Der interne Fehlerzähler ist nur im Debug-Modus zugänglich:

```
uint8_t rotmess_err(){//Fehlerabfrage
  if (rotmess_err_ct){//wenn Fehler aufgetreten
                      //sind
    rotmess_err_ct=0; //Fehlerzähler löschen
    return 1;         //Rückkehr mit 1 (wahr)
  }                   //sonst
  return 0;           //Rückkehr mit 0 (falsch)
}
```

Das Testprogramm »test_rotmess«

Das Testprogramm bindet außer »rotmess.h« folgende Header ein:

```
#include "comir_pc.h"//PC-Eingabe und -ausgabe
#include "pwm.h"     //Geschwindigkeitssteuerung
```

Vom PC wird auf ein 6-Byte-Datenpaket gewartet, die Motoren bewegt und ein 8-Byte-Paket zurückgesendet (in »comir_pc.h«):

```
#define COM_PC_RMSG_LEN  6
#define COM_PC_SMSG_LEN  8
```

Empfangsdaten:

- Byte 1 und 2: Pulslänge Motor R (OCR5B),
- Byte 3 und 4: Pulslänge Motor L (OCR5C),
- Byte 5 und 6: Pulsperiode Motor R und L (OCR5A).

Zurückgesendete Bytes:

- Byte 1 und 2 bzw. 5 und 6: empfangene Bytes 1 und 2 bzw. 3 und 4.
- Byte 3 und 4 bzw. 7 und 8: Winkelschritt pro s Motor R bzw. L.

Variablen des Hauptprogramms:

```
uint8_t rmsg[COM_PC_RMSG_LEN];
uint8_t smsg[COM_PC_SMSG_LEN];
int main(){
  int16_t speed_R, speed_L, pwm;
  uint8_t state=0; //Programmzustand
```

Treiberinitialisierung, globale Interrupt-Freigabe:

```
  rotmess_init(); //initialisieren aller Treiber
  com_pc_init();
  pwm_init();
  sei();           //Interrupts einschalten
  while(1){ ...  }
```

Das Hauptprogramm ist ein Zustandsautomat:

- Zustand 0: Warte auf 6-Byte-Nachricht vom PC,
- Zustand 1 und 2: Bewegung ohne Messung,
- Zustand 3: Bewegung mit Messung der Winkelschritte,
- Zustand 4: Messergebnisse zum PC senden.

Im Zustand 0 wird auf eine 6-Byte-Nachricht vom PC gewartet. Falls keine da ist, wird die PWM angehalten:

```
if (state == 0){
 if (com_pc_get(rmsg)){//wenn neue Nachricht
  pwm = rmsg[0]<<8 | rmsg[1];
  pwm_set_R(pwm); //Wert für Motor R einstellen
  pwm = rmsg[2]<<8 | rmsg[3];
  pwm_set_L(pwm); //Wert für Motor L einstellen
  OCR5A = rmsg[4]<<8 | rmsg[5];
  pwm_start();       state = 1;
 }
 else
  pwm_stop();
 }
```

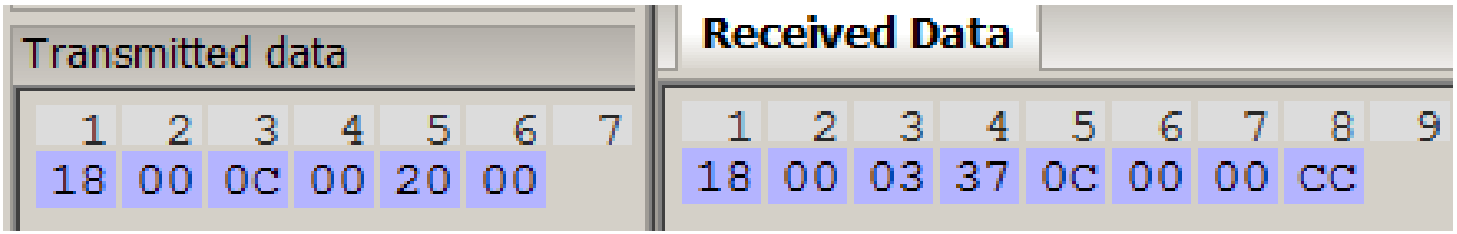
In den Zuständen 1 bis 3 passiert nur etwas, wenn neue Winkelmessdaten bereit sind, d.h. alle 1 s. In Zustand 2 und 3 soll sich eine konstanten Geschwindigkeit einstellen. Im Zustand 4 werden die PWM-Vorgaben und Zählwerte zum PC gesendet und der Zustand auf null zurückgesetzt:

```
if (state && rotmess_get(&speed_R, &speed_L)){
 state++;              //nach jeder Messung Zustand++
 if (state>3){         //3. Messergebnis zum PC senden
  smsg[0] = rmsg[0];     smsg[1] = rmsg[1];
  smsg[2] = speed_R>>8; smsg[3] = speed_R & 0xff;
  smsg[4] = rmsg[2];     smsg[5] = rmsg[3];
  smsg[6] = speed_L>>8; smsg[7] = speed_L & 0xff;
  com_pc_send(smsg);
  state=0;
 }
}
```

Treiber »rotmess« ausprobieren

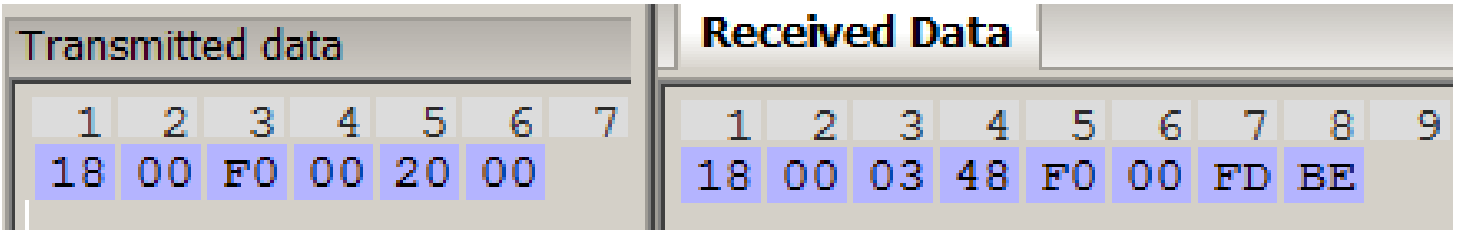
- Hardware-Aufbau siehe Folie 7.
- PmodUSBUSART an JH oben und USB-Verbindung zum PC.
- JHX und JLX auf »gekreuzt (=)«.
- Projekt »F11-rotmess\rotmess« übersetzen und starten.
- HTerm starten. 8N1 9600 Baud. Connect.

Testbeispiel mit HTerm:



PWM_R	speed_R	PWM_L	speed_L
$\frac{0x1800}{0x2000} = 75\%$	$\frac{0x337}{228} = 3{,}61\,\frac{\mathrm{U}}{\mathrm{s}}$	$\frac{0x0C00}{0x2000} = 37{,}5\%$	$\frac{0x0CC}{228} = 0{,}89\,\frac{\mathrm{U}}{\mathrm{s}}$

Weiteres Testbeispiel mit HTerm:

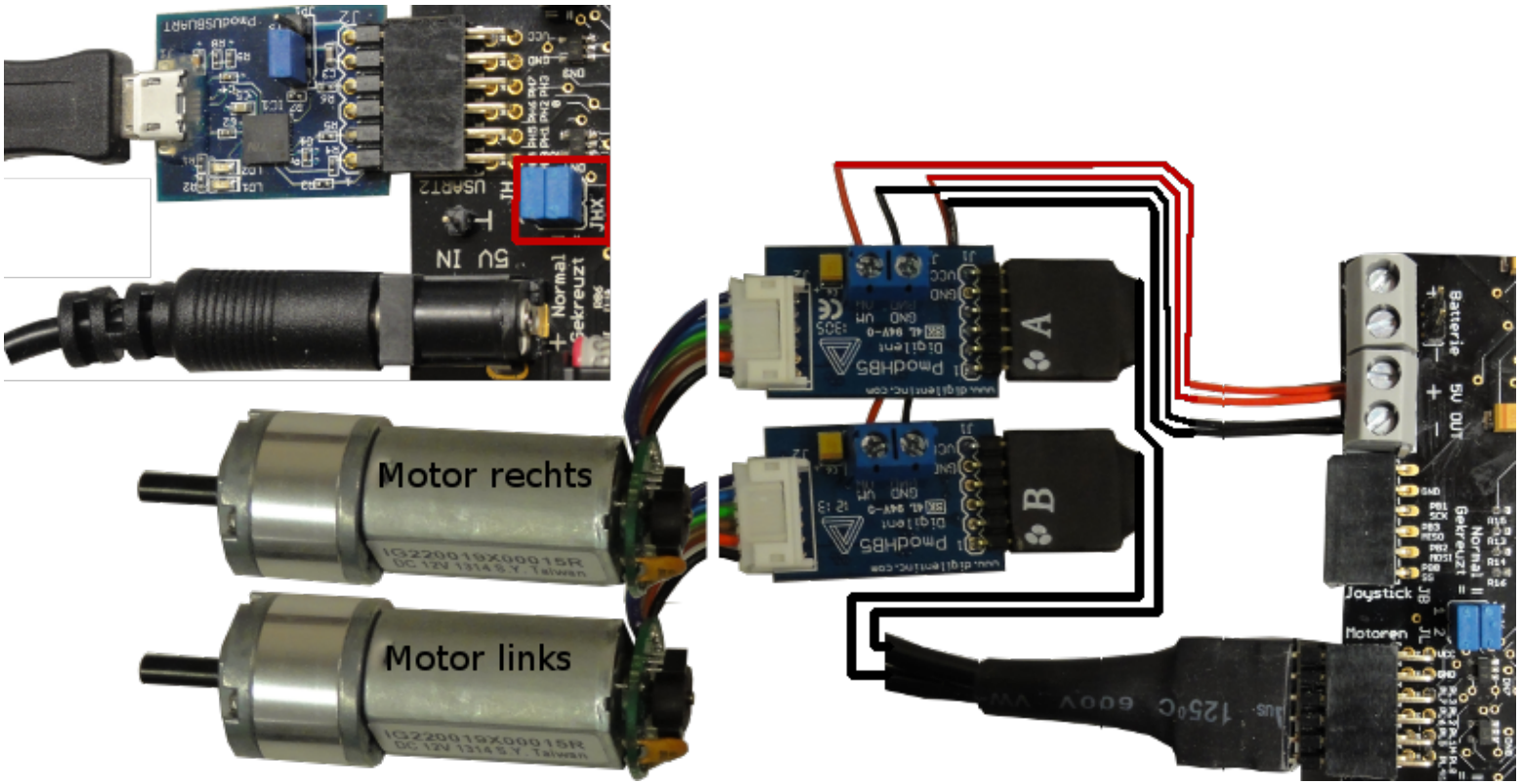


PWM_R	speed_R	PWM_L	speed_L
$\frac{0\text{x}1800}{0\text{x}2000} = 75\%$	$\frac{0x348}{228} = 3{,}68\,\frac{\text{U}}{\text{s}}$	$\frac{-0\text{x}1000}{0\text{x}2000} = -50\%$	$\frac{-0x242}{228} = -2{,}54\,\frac{\text{U}}{\text{s}}$

Absolute Pulsweite in den Beispielen :$0\text{x}2000/8\,\text{MHz} \approx 1\,\text{ms}$

Aufgaben

Aufgabe 11.1: Testbeispiele aus der Vorlesung



Anschluss Motoren, PModUSBUSART, USB-Kabel zum PC, ...:

1. Treiber »pwm« entsprechend Folie 18 ausprobieren,
2. Treiber »rotmess« entsprechend Folie 37 ausprobieren.

Aufgabe 11.2: Fahrzeugsteuerung

1. Bauen Sie das Fahrzeug auf mit
   - H-Brücken mit Motoren wie auf Folie 7 an JL und
   - Bluetooth-Modul wie auf Foliensatz/Handout 6 an JE.
2. Ersetzen Sie im Testprogramm »test_pwm« ab Folie 16 den Treiber für die Kabelverbindung an USART2 durch einen funktionsgleichen Bluetooth-Treiber an USART0. (Erfordert nur den Ersatz von USART2 durch USART0.)
3. Testen Sie das Fahrzeug mit HTerm-Eingaben.
4. Erweiteren Sie das Programm so, dass die Restfahrzeit von jedem HTerm-Fahrkommando binär auf den LEDs an Port J angezeigt wird.

Aufgabe 11.3: Fahrzeugsteuerung über Python

1. Erweiteren Sie das Mikrorechnerprogramm »test_pwm.c« so, dass nach Empfang und Verarbeitung von jedem 6-Byte-Datenpaket das Byte 0xFF zurückgesendet wird.
2. Schreiben Sie ein Python-Programm, das über die serielle Schnittstelle an das Mikrorechnerprogramm eine Folge von Tupeln:

   $$(\text{PWM\_R}, \text{PWM\_L}, \text{Dauer\_in\_0,1s-Schritten})$$

   sendet. Das erste Tupel ist sofort zu senden und jedes weitere erst nach Empfang des Quittungsbytes 0xFF für die Abarbeitung des vorherigen Tupels.
3. Entwickeln Sie Testbeispiele für unterschiedliche abzufahrende Bahnen, bei denen das Fahrzeug am Ende wieder etwa auf der Startposition in Startrichtung ankommt.

Aufgabe 11.4: Inkrement-Aufzeichnung

Erweiteren Sie das Mikrorechnerprogramm aus Aufgabe 11.3 dahingehend, dass das Programm während der Bewegung alle Inkrement-Tupel

$$(\Delta w_\mathrm{R}, \Delta w_\mathrm{L}) \in \{(-1,-1), (-1,0), (-1,1), (0,-1), (0,1), (1,-1), (1,0), (1,1)\}$$

kodiert als Hex-Ziffern kleiner 0xF (zwei Werte je Byte) an den PC sendet. Das Python-Programm soll zusätzlich alle empfangenen Byte-Werte $\neq 0\mathrm{xFF}$ als hex-Ziffernfolge auf die Konsole ausgeben. Leiten Sie die Konsolenausgabe mit »PythonProgramm > Datei« zur späteren Auswertung in eine Datei um.
Alternativ zur Nutzung der Programme aus der vorherigen Aufgabe kann die Tupelfolge

$$(\mathrm{PWM\_R}, \mathrm{PWM\_L}, \mathrm{Dauer\_in\_0{,}1s\text{-}Schritten})$$

auch als Konstante in das C-Programm compiliert werden.

Aufgabe 11.5: Inkrement-Aufzeichnung

Schreiben Sie in Fortsetzung von Folie 24 ein Programm zur Bestimmung der 2,5D-Fahrzeugposition (x, y, α) relativ zur Startposition und Richtung $(x, y, \alpha)_0 = (0, 0, 0)$, indem für jedes aufgezeichnete Tupel $(\Delta w_\mathrm{R}, \Delta w_\mathrm{L}) \neq (0, 0)$ der Folgewert von (x, y, α) berechnet wird.

Hinweis: Schreiben Sie das Programm zuerst in Python zur Berechnung der abgefahrenen Bahn aus einer aufgezeichneten Folge von Inkrement-Tupeln. Verwenden Sie als Testbeispiele Bahnen, bei denen das Fahrzeug am Ende nahe der Startposition in Startrichtung ankommt. Programmieren Sie nach Vorlage des getesteten Python-Programms ein Programm, das die 2,5D-Bahn auf dem Mikrorechner bestimmt. Verwenden Sie im Mikrorechnerprogramm für Positions- und Winkelwerte Gleitkommazahlen (float).